Coupled Multi-Physics Approach in CESAR

Timothy J. Tautges, Vijay Mahadevan, Rajeev Jain, Tom Peterka
Mathematics and Computer Science Division
Argonne National Laboratory

Exascale Research Conference
Washington, DC
October 3, 2012

Outline

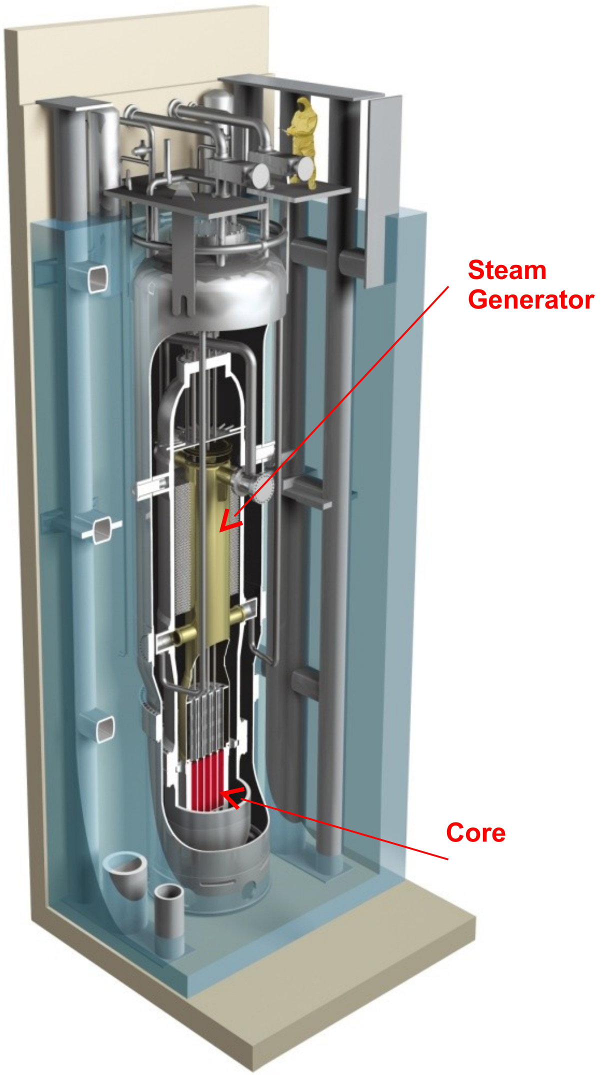
- Introduction & Motivation
- Physics
- Coupling approaches
- Mesh-related details
- Results so far
- Conclusions

Introduction: Nuclear Reactor Physics

- Nuclear reactor core designs vary by coolant, neutron spectrum, fuel types
  - “Light water” reactors: low-Z coolant/moderator, low-energy neutrons
  - “Fast” reactors: higher-Z coolant, higher-energy neutrons
- Interaction between physics strongly affects fission reactivity
  - LWR: coolant boiling, strong fuel doppler affect (oxide)
  - Fast reactor: structural expansion, weak fuel doppler affect (metal)
- Safety/performance of reactors inherently depends on coupling between physics
- Localized 3D effects, e.g. near control blades or grid spacer structures, affects safety & reliability, motivating need for much higher fidelity than current homogenized methods

Reactor Simulation is an Exascale Computing Problem

- Current generation of nuclear reactors designed by experiment first, computing second
  - We can no longer do this for next-generation reactors
- Neutron transport, thermal/fluid transport with turbulence, structural mechanics are each petascale+ computing problems
  - To get the right answer, we need to do all 3, coupled



Coupled Neutron, Fluid, Heat Transport

Neutron transport:

$$\hat{\Omega}\cdot\vec{\nabla}\Psi+\Sigma_t(E)\Psi=\iint\Sigma_s(E')\Psi(\hat{\Omega}',E')d\hat{\Omega}'dE'+\mathrm{X}(E)\int dE'\,\nu\Sigma_f(E')\int d\hat{\Omega}'\Psi(\hat{\Omega}',E')$$

in-scattering *fission*

Heat transport:

$$\rho C_p u\cdot\nabla T-\nabla\cdot(k\nabla T)=q_{vol}-q_{surf}$$

Fluid transport:

$$\rho\vec{u}\cdot\nabla\vec{u}=-\nabla p+\frac{1}{\mathrm{Re}}\nabla^2\vec{u}$$

Coupled Neutron, Fluid, Heat Transport

$$\hat{\Omega}\cdot\vec{\nabla}\Psi + \Sigma_t(E)\Psi = \iint \Sigma_s(E')\Psi(\hat{\Omega}',E')\,d\hat{\Omega}'\,dE' + \underbrace{\mathrm{X}(E)\int dE'\,\nu\Sigma_f(E')\int d\hat{\Omega}'\,\Psi(\hat{\Omega}',E')}$$

rho, T-dependent

fission heating

$$\rho C_p u\cdot\nabla T - \nabla\cdot\left(k\nabla T\right) = q_{vol} - q_{surf}$$

$$\rho\vec{u}\cdot\nabla\vec{u} = -\nabla p + \frac{1}{\mathrm{Re}}\nabla^2\vec{u}$$

Coupled Neutron, Fluid, Heat Transport

Code1 (UNIC or proxy)

$$\hat{\Omega}\cdot\vec{\nabla}\Psi + \Sigma_t(E)\Psi = \iint \Sigma_s(E')\Psi(\hat{\Omega}',E')\,d\hat{\Omega}'\,dE' + \mathrm{X}(E)\int dE'\,\nu\Sigma_f(E')\int d\hat{\Omega}'\,\Psi(\hat{\Omega}',E')$$

rho, T-dependent

fission heating

$$\rho C_p u\cdot\nabla T - \nabla\cdot\left(k\nabla T\right) = q_{vol} - q_{surf}$$

$$\rho\vec{u}\cdot\nabla\vec{u} = -\nabla p + \frac{1}{\mathrm{Re}}\nabla^2\vec{u}$$

Code 2 (Nek5000 or proxy)

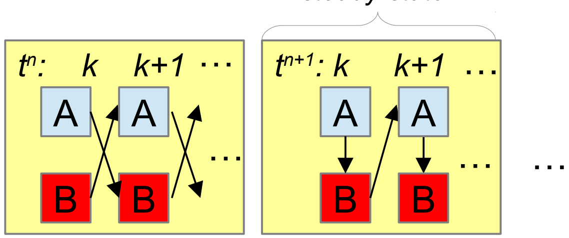
Full/Original Physics Codes

	Nek5000	UNIC
Physics	Incompressible NS	Boltzmann transport
Discretization	SEM w/ LES turb (NxNxN GLL basis)	FEM (linear, quadratic)
Solver	Native semi-implicit with AMG	3-level hierarchy (eigenvalue, energy, space/angle), with PETSc for space/angle
Materials, BCs	User-defined functions	ExodusII-like element blocks, sidesets
Mesh type	Ucd hex	Ucd hex, tet, prism
Implementation	F77 + C, 100k lines	F90, 260k lines
Mesh, data storage	Common blocks	F90 modules
Scalability	2000 Gorden Bell prize, 71% strong scaling on 262k cores	2009 Gordon Bell finalist, 76% strong scaling on 295k cores
Effort invested	~30 man-years	~10 man-years

Coupling Approaches

- Different flavors of coupling schemes have variations in stability, accuracy, and software characteristics

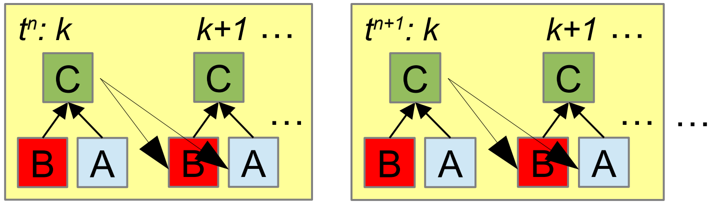
Loose:



Jacobi *Gauss-Seidel*

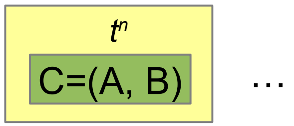
- Separate/original physics components (A, B)
- Operator-split solution, Jacobi or Gauss-Siedel
- Conditionally stable
- Convergence may be slow, depending on relative magnitude/variations of coupling terms

Tight:



- Separate/original physics components (A, B)
- Physics compute Jacobian & residual, driver (C) computes new solution
- Can be unconditionally stable
- Convergence is better than for loose coupling
- May require physics component modifications to return Jacobians/residuals

Full:



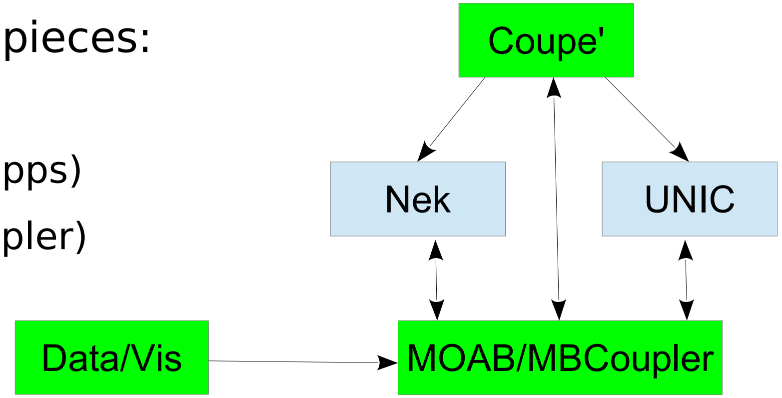
- All equations solved simultaneously, in same coupled system
- No iterations between physics required
- Physics components must be combined, solution procedures re-written

Coupling Considerations *For CESAR*

- For a variety of reasons, full coupling approach isn't the best choice
  - Difficult to express T-dependence of nuclear cross sections in differentiable form, making it difficult to incorporate in unified non-linear solution procedure
  - Neutron transport may not even be expressed in PDE form (e.g. Monte Carlo), difficult to formulate Jacobian
  - Difficult to develop single-physics module that performs at exascale, let alone a fully coupled multi-physics code
- Structuring our coupled code work to allow simultaneous investigation of multiple (loose and tight) coupling approaches
  - Model coupled system as solution of individual physics (possibly in original code modules) and explicit solution transfer for coupling terms

Coupling Considerations *For CESAR*

- For a variety of reasons, full coupling approach isn't the best choice
  - Difficult to express T-dependence of nuclear cross sections in differentiable form, making it difficult to incorporate in unified non-linear solution procedure
  - Neutron transport may not even be expressed in PDE form (e.g. Monte Carlo), difficult to formulate Jacobian
  - Difficult to develop single-physics module that performs at exascale, let alone a fully coupled multi-physics code
- Structuring our coupled code work to allow simultaneous investigation of multiple (loose and tight) coupling approaches
  - Express coupled solution as solution of individual physics (possibly in original code modules) and explicit solution transfer for coupling terms
  - Compose coupled code from 3 primary pieces:
    - Driver (Coupe')
    - Physics modules (UNIC, Nek, or mini-apps)
    - Solution transfer tool (MOAB & MBCoupler)



Coupe' Coupled Physics Driver

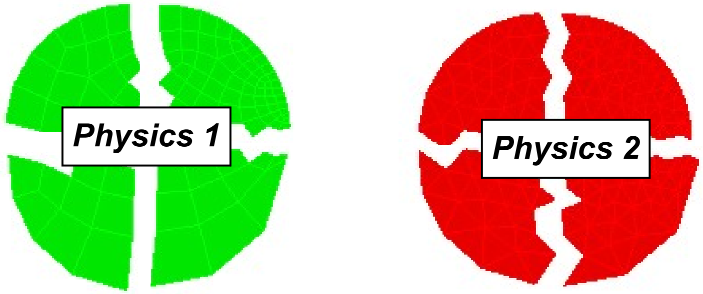
- Designed to allow both loose (Operator-Split) and tight coupling strategies
  - Tight coupling better resolves spatio-temporal coupling, but requires more from individual physics (residuals, etc.)
  - Run-time selection of coupling scheme with other parameters fixed allows apples-apples comparisons
- Coupe' coordinates:
  - Iterations in a timestep and over timesteps
  - Calling solution transfer
  - Testing for convergence
- Coupe' framework has few minimal requirements on individual physics from a software perspective:
  - Store corresponding mesh information to or read mesh info from MOAB
  - Push/pull coupled fields to/from MOAB tags
  - Provide functions that handle creation, destruction, setup, solve
  - Optionally, provide function for nonlinear residual
  - Optionally, provide the action of a preconditioner on a vector to use as an accelerator in the coupled system solve

Mesh-Oriented datABase (MOAB)

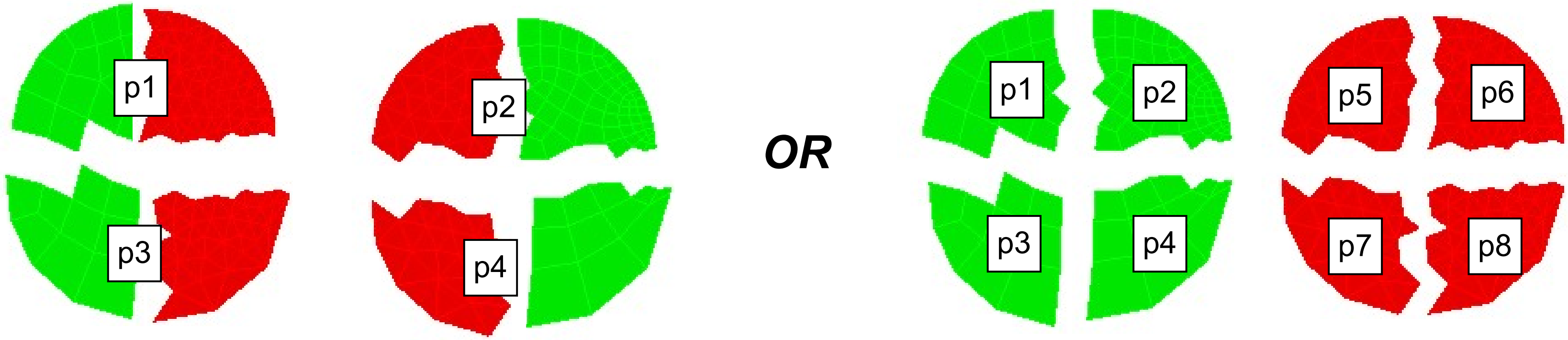
- Library for representing, manipulating structured, unstructured mesh models
- Supported mesh types:
  - FE zoo (vertices, edges, tri, quad, tet, pyramid, wedge, knife, hex)
  - Polygons/polyhedra
  - Structured mesh
- Optimized for memory usage first, speed second
- Implemented in C++, but uses array-based storage model
- Mesh I/O from/to various formats
  - HDF5 (custom), vtk, CCMIO (Star CD/CCM+), Abaqus, CGM, Exodus
- Main parts:
  - Core representation
  - Tool classes (skinner, kdtree, OBBtree, ParallelComm, …)
  - Tools (mbsize, mbconvert, mbzoltan, mbcoupler, …)
- Parallel model supports typical element-based decompositions, with typical mesh-based functions (shared interface, ghost exchange, ownership queries)

MOAB-Based Solution Transfer

- ***Meshes:*** Each physics type is solved on an ***independent mesh*** whose characteristics (element type, density, etc.) is most appropriate for the physics



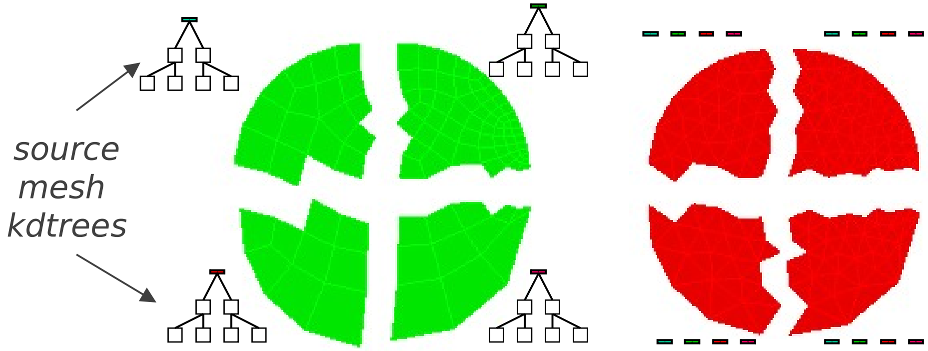
- **Distribution:** Each physics type and mesh is ***distributed independently*** across a set of processors, defined by an MPI communicator for each mesh



- **Implementation:** On a given processor, all meshes are stored in a ***single iMesh instance***, and that instance communicates with all other processors containing pieces of any of those meshes.

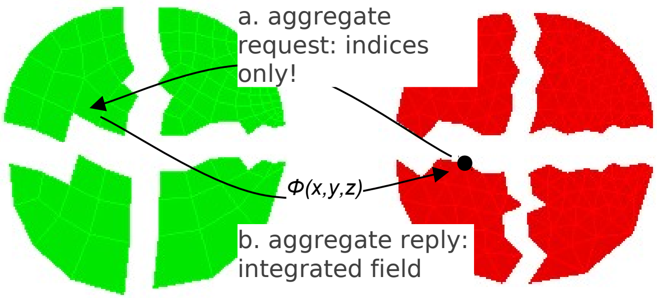
Solution Transfer: 4 Steps

1. Initialization

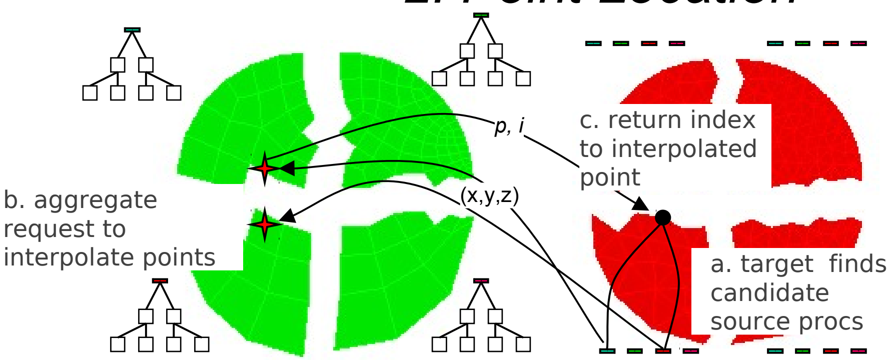


*target procs
store
all kdtree roots*

3. Interpolation



2. Point Location



i: (x, y, z), h, (u, v, w)
...

Source proc: index of mapped points:
Target position, local element handle,
param coords

h, p, i
...

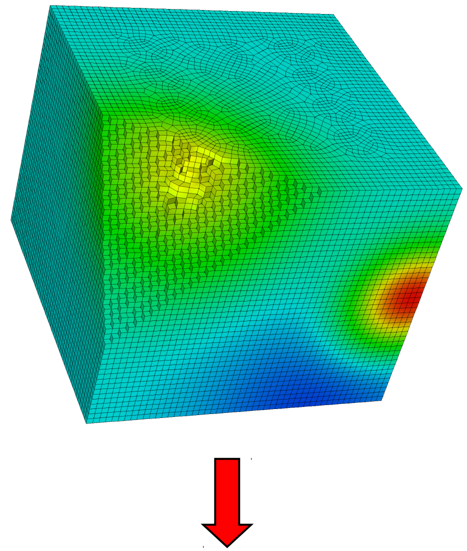
Target proc: local handle, source proc,
remote index

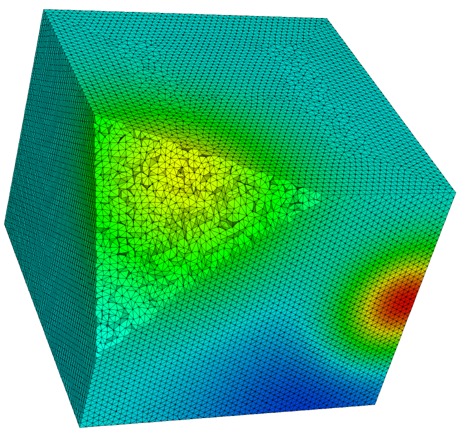
4. Normalization

- Minimize data transferred
  - Store index close to source field, communicate indices only
- All communication aggregated, using “crystal router” for generalized all-to-all

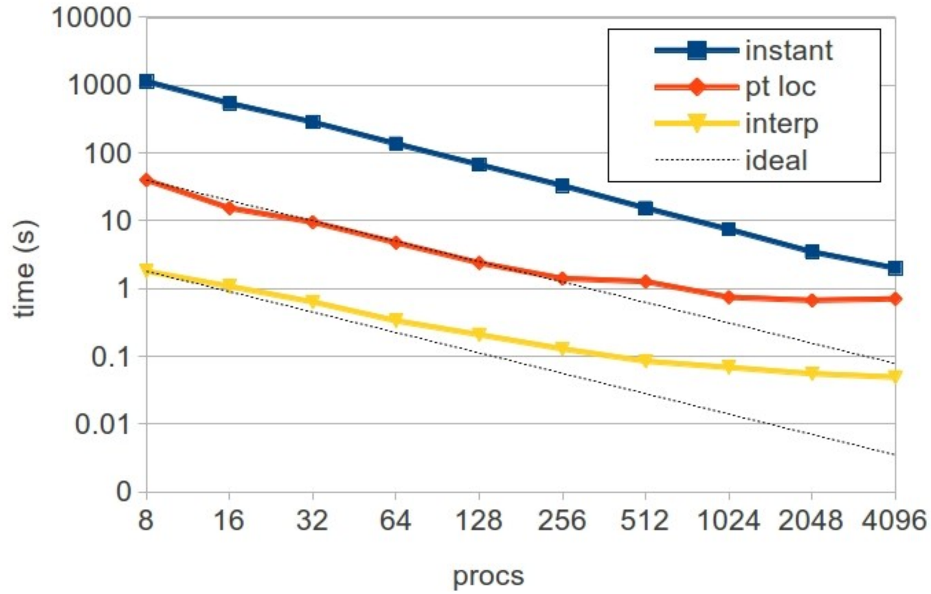
Solution Transfer: Performance, Accuracy

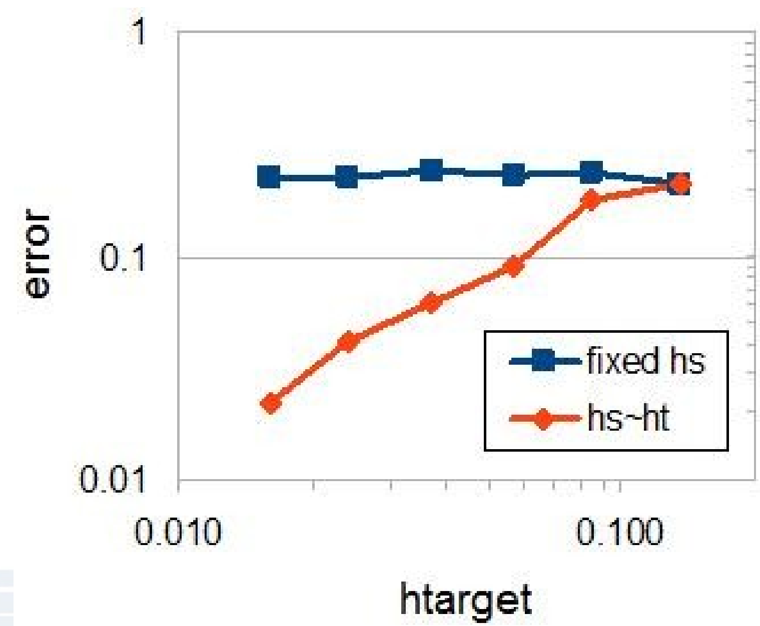
7M Hexes





28M Tets





Exascale Issues

- Partitioning physics over processors
- Parallel solution transfer
- Local tree search
- Memory sharing

Solution Transfer: Distribution Over Processors

- Assuming fixed number of procs and fixed (possibly non-equal) problem sizes for physics, 2 choices for partitioning physics solutions over machine
- Homogeneous: each proc solves a piece of each physics
  - Requires good strong scaling of each physics
  - Can do both Jacobi- and Gauss-Siedel-type loose coupling
  - Easier load balancing, even with sub-cycling in time
- Disjoint: each physics solved on set of procs disjoint from other physics procs
  - Lighter strong scaling requirements
  - Gauss-Siedel scheme leaves processor sets idle, Jacobi requires accurate prediction of runtime
- Our approach: don't over-constrain any of the underlying support (i.e. solution transfer can support both homogeneous and disjoint scenarios)

Solution Transfer: Mesh Search Details

- Current parallel search method does linear search over top-level boxes on each proc, which is both scalability and memory problem
- Change to a rendezvous-type method, where intermediate set of procs with deterministic partition of overall bounding box & intersecting processor boxes directs packets to correct proc(s)
- Local search tree currently a kdtree, but probably more efficient to use a bvh tree
  - Tree search consists of tree traversal (cheap), in-leaf element query (expensive); bvh adds tree complexity to reduce leaf complexity
- In process of implementing/testing bvh tree
- Will implement rendezvous method in early FY13

Memory Sharing Between Physics, MOAB

- MOAB uses array-based storage of most “heavy” data, and exposes API functions giving access to contiguous chunks of those data (mesh definition & mesh-based variables)

```
Range::iterator iter = myrange.begin(); int count; double *data;
while (iter != myrange.end()) {
  tag_iterate(tag_handle, iter, myrange.end(), count, (void*&)data_ptr );
  iter += count;
}
```

- Small applications show that this almost completely eliminates API cost for accessing variable data memory owned by MOAB
- Advantages:
  - Eliminates memory copy between physics & backplane, saving memory and time
  - Allows direct use of parallel services like I/O, in-situ viz
  - Simplifies workflow (pre, analysis, post) because no issues with data formats for various physics
  - Will allow faster transition to memory manipulations for manycore, GPU
- The fine print: depends heavily on mesh, DOF ordering in physics

Coupling/Datavis Proxy Application

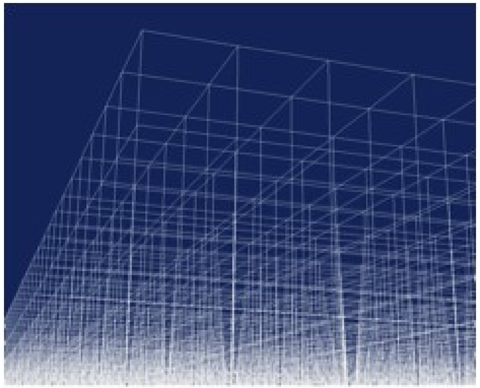
- Purpose of proxy application: study specific aspect(s) of exascale problem
  - Compute proxy: computation/communication kernel(s), with narrow option/feature set
  - Coupling/datavis: data-intensive, exercising overall simulation workflow
- Goals
  - Provide shared, reference implementation for calling data-oriented tasks needed by most proxy activities
  - Acquire representative raw data in the coupled application interface & make available to coupling/datavis proxies
  - Test & benchmark various options of data-oriented services
  - Use results to influence co-design of hardware *and software (libraries)*
  - Make proxies available to other subgroups (eg. GPGPUs, programming models, performance modeling, library developers)
- Factors
  - How well do proxies represent actual problems (in our case, real data)?
    - Real features, eg., vortices, data distributions, outliers, correlations among variables
  - Difficult if not impossible to generate without running actual solvers
    - However, results can be stored and later read back into proxies without rerunning simulation

Coupling/Datavis Proxy Approach

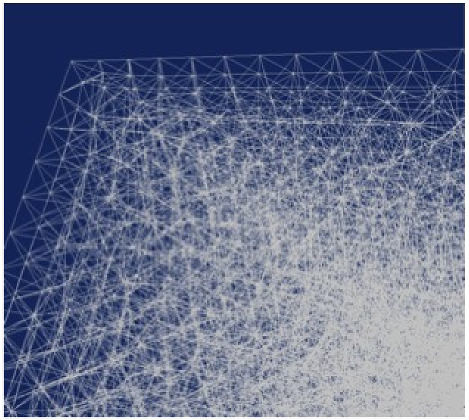
- Construct single high-level proxy shell implementing:
  - Option processing (proxy type, mesh/data files, other parameters)
  - library initialization (MOAB, vtk, maybe solvers)
  - Coordination of I/O
- Individual options/functions for studying specific things:
  a) Coupling & solution transfer
  b) Data analysis
  c) Visualization
  d) Storage

(a) Coupling & Solution Transfer Proxy

- Questions/issues
  - Performance, accuracy of solution transfer
  - Partition of multiple meshes over processors
  - Hybrid programming models & partitioning over cores/nodes
  - Solution of post-transfer numerical constraints (conservation/normalization)
- Method
  - Reads two different test meshes
  - Transfers solution from first mesh to second
  - Compute error for analytic solution data
  - Measure performance/accuracy



Sample mesh of 1M hexahedral cells.



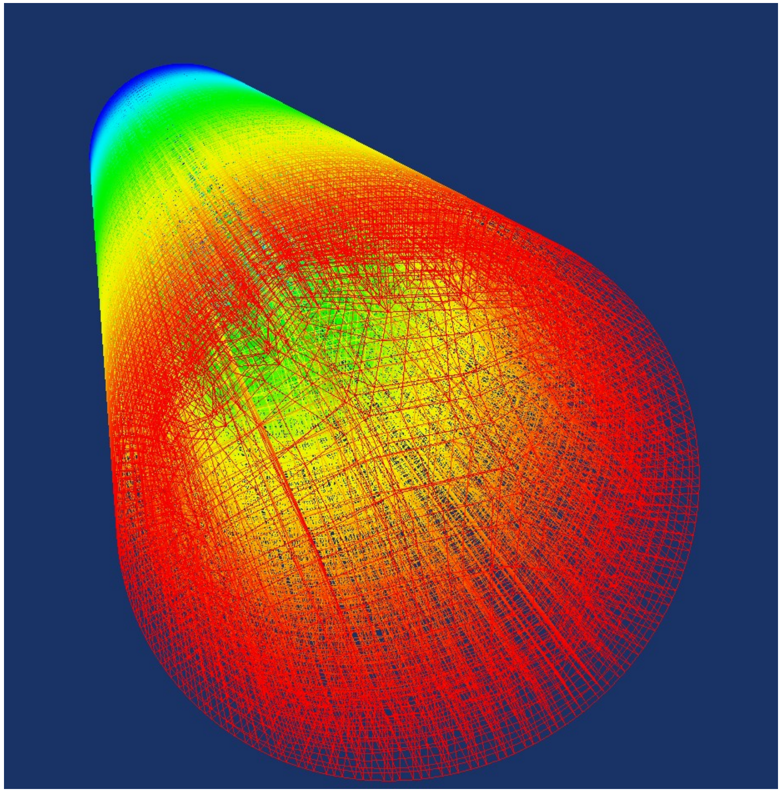
Sample mesh of 1M tetrahedral cells.

(b) Data Analysis Proxy

- Questions/Issues
  - Size/data model of derived data
  - Placement and scheduling of analysis tasks
  - Resampling, interpolating, subsettting are valid operations
- Method
  - Takes mesh interface handle
  - Derive/compute new analysis data (e.g. Lambda-2 vorticity)

(c) Visualization Proxy

- Questions/Issues
  - Memory savings vs. code complexity from in-situ visualization
  - Placement and scheduling of visualization work
- Method
  - Takes mesh interface handle
  - Call in-situ rendering of various types (polygon, volume rendering)
  - Assess resource usage with memory sharing vs. deep copy



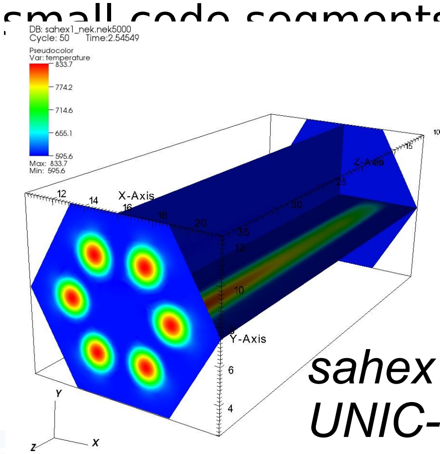
5K 4th-order spectral element mesh with proxy CFD solution data, rendered in wireframe mode.

(d) Storage Proxy App

- Questions/Issues:
  - Checkpoint reading/writing performance
  - Analysis data vs. checkpoint data?
  - Storage-side format & layout affects on scaling
- Method
  - Takes mesh interface handle
  - Writes mesh to storage (checkpoint)
  - Writes results of analysis or visualization proxy to storage
  - Compare performance with I/O benchmarks and published performance for similar I/O workloads

Towards Full-Up Exascale Coupled Reactor Analysis

- Proxy apps should inform both hardware/middleware design as well as various aspects of full-up exascale applications
- Physics (computation, communication)
  - Performance of key mat-mat multiply kernel, scalability of pressure solve & MOC methods, preconditioning affects on multi-layer iteration (eigenvalue, energy, space), deep memory hierarchy design
- Coupling/datavis (data size, complexity)
  - Support for exascale-class datasets across software stack
- In many cases, improvements from proxies can directly inform full-up codes
  - Physics: many kernels isolated in relatively small code segments
  - Libraries: same ones being used for proxies
- Leveraging work on coupled reactor analysis efforts from NEAMS targeted at science problem on smaller machines



sahex1 test
UNIC-Nek5000

Conclusions

- Coupled multi-physics comes in many forms; for reactor analysis, loose, (various forms of) tight coupling hold the most promise
  - Key research focus, interactions between scalability and numerics
- Co-design for coupling, and esp. datavis, involves not only computation/communication kernels, but also software stack co-design for exascale-class data
  - Lots more moving (software) parts
- Proxy apps for this area are as much about data as computation/communication
  - Layout important, with added complexity of libraries & metadata
  - In a sense, proxy app is closer to full-up app, since libraries are the same